AngularJS Fundamentals

by Jeremy Zerr

Blog: http://www.jeremyzerr.com
LinkedIn: http://www.linkedin.com/in/jrzerr
Twitter: http://www.twitter.com/jrzerr

What is AngularJS

- ☐ Open Source Javascript MVC/MVVM framework.
- ☐ Helps manage complexity of your client-side Javascript code.
- ☐ Extend HTML vocabulary with your own elements and attributes.
- ☐ Built-in HTTP and more importantly REST API web service integration.

What is AngularJS

- ☐ Includes its own client-side templating language.
- ☐ Two-way data binding to allow the Model to be used as the source of truth.
- ☐ Has client-side routing to enable creating single page apps (SPA)
- ☐ Sponsored by Google, lots of contributions by Google employees, but on Github as public repo.

What Can You Create With AngularJS?

- [ ] Web Apps
- [ ] Mobile Apps (Ionic)
- [ ] Desktop apps (Chrome App, Electron)

AngularJS Examples

Starter AngularJS app

- ☐ We'll start by defining some Javascript objects in a controller to represent our Models
- ☐ Then we will display the object using AngularJS built-in template system
- ☐ Follow along with this Plunker

Starter AngularJS app (templating)

```
<h1>Starter AngularJS app</h1>
<div ng-controller="ToddlerCtrl">
  <h2>Toddlers</h2>
  <table>
    <tr>
      <th>Name</th>
      <th>Birthday</th>
      <th>Happy?</th>
    </tr>
    <tr ng-repeat="toddler in toddlers">
      <td>{{toddler.name}}</td>
      <td>{{toddler.birthday}}</td>
      <td>{{toddler.happy}}</td>
    </tr>
  </table>
</div>
```

- This shows what AngularJS client-side templating looks like
- ng-controller provides an identifier to link with code
- ng-repeat iterates through a variable in the controller's $scope

Starter AngularJS App (controller)

```
1  // Instantiate the app, the 'myApp' parameter must
2  // match what is in ng-app
3  var myApp = angular.module('myApp', []);
4
5  // Create the controller, the 'ToddlerCtrl' parameter
6  // must match an ng-controller directive
7  myApp.controller('ToddlerCtrl', function ($scope) {
8
9    // Define an array of Toddler objects
10   $scope.toddlers = [
11     {
12       "name": "Toddler One",
13       "birthday": "1/1/2011",
14       "happy": true
15     },
16     {
17       "name": "Toddler Two",
18       "birthday": "2/2/2011",
19       "happy": true
20     },
21     {
22       "name": "Toddler Three",
23       "birthday": "3/3/2011",
24       "happy": false
25     }
26   ];
27
28 });
```

- ☐ The name of our app is myApp
- ☐ Controller is ToddlerCtrl
- ☐ We define the controller and fill our scope up with Toddler objects. (its just a Javascript data structure - JSON)

What is a $scope?

- [ ] $scope is the application ViewModel
- [ ] It is the glue between the controller and the view.
- [ ] AngularJS documentation on $scope

Using client-side models from different data sources

- Data sources:
  - Using JSON in the initial page load to pre-populate Services
  - Using a static JSON file
  - Using a REST API
- We'll build on the previous example by creating our Models using different data sources.
- Follow along with this Plunker

Using JSON in initial page load

```
<script type="text/javascript">
// Define an array of Toddler objects
window.toddlers = [
  {
    "name": "Toddler One",
    "birthday": "1/1/2011",
    "happy": true
  },
  {
    "name": "Toddler Two",
    "birthday": "2/2/2011",
    "happy": true
  },
  {
    "name": "Toddler Three",
    "birthday": "3/3/2011",
    "happy": false
  }
];
</script>
```

- ☐ Assign the JSON to a variable in a <script> tag.
- ☐ Create a Service using $resource
- ☐ Pass the JSON to the constructor

```
myApp.factory('Toddler', function($resource) {
  return $resource('toddlers.json');
});
```

```
$scope.toddlers = [];
angular.forEach(window.toddlers, function (item) {
  $scope.toddlers.push(new Toddler(item));
});
```

Using REST API & JSON file

```
28 /**
29  * Adult is a service that calls a REST API
30  * It's not really a REST API, but just calling our local .json file
31  * as an example
32  * If you call Adult.query(), it will GET adults.json
33  * If you call Adult.get({}, {aid: 1}) it will GET adults/1.json
34  */
35 myApp.factory('Adult', function($resource) {
36   return $resource('adults/:adultId.json', {adultId: '@aid'});
37 });
38
39 // Create the controller, the 'PersonCtrl' parameter must
40 // match an ng-controller directive
41 myApp.controller('PersonCtrl', function ($scope, Toddler, Teen, Adult) {
42
43   // Initialze Toddlers from JSON defined on initial page load
44   $scope.toddlers = [];
45   angular.forEach(window.toddlers, function (item) {
46     $scope.toddlers.push(new Toddler(item));
47   });
48
49   // Teens are from static json file
50   $scope.teens = Teen.query();
51
52   // Adults are from REST API
53   $scope.adults = Adult.query();
54
55   // Example of grabbing single Adult from REST API
56   $scope.singleAdult = Adult.get({adultId: 1});
57
58 });
```

- You create a URL template. This identifies the object ID field (aid)
- Controller body shows initializing data 4 different ways
- With the same Adult resource, you do a get() to request a single object

Templating methods

- ☐ Directives + AngularJS templating allows you to create custom HTML markup, both elements and attributes
- ☐ Templating types:
  - ☐ We've already seen inline HTML
  - ☐ Can define within Javascript
  - ☐ Can include within <script> tags
  - ☐ Can include in an external HTML file
- ☐ We'll take our existing code, pick the local JSON file as the data source, and show a comparison between these different templating methods.
- ☐ Follow along with this Plunker

Templating method: Javascript

```
/**
 * Use a javascript-based template
 */
myApp.directive('teenJavascript', function() {
  return {
    restrict: 'AE',
    scope: {
      teen: '='
    },
    template: '<tr><td>{{teen.name}}</td><td>{{teen.birthday}}</td><td>{{teen.happy}}</td></tr>'
  };
});
```

- ☐ We use an AngularJS directive
- ☐ Can also declare your template right in Javascript

Templating method: <script>

```
<script type="text/ng-template" charset="utf-8" id="teen-internal.html">
  <tr>
    <td>{{teen.name}}</td>
    <td>{{teen.birthday}}</td>
    <td>{{teen.happy}}</td>
  </tr>
</script>
```

- Template cache can be pre-loaded by including a template within <script> tags, using a special type.
- Any directive that references teen-internal.html first looks in template cache before requesting the html file from the server.

Two-way data binding

- You can bind a variable in $scope to elements or inputs
- You can also use a $scope variable to control which CSS class gets applied to an element
- If the input changes its value, the underlying $scope variable also changes.
- Follow along with this Plunker
- How would you have done this with jQuery + Mustache? See this Plunker

Two-way data binding: ng-model + ng-class

```
<tr ng-repeat="teen in teens" ng-class="{'happy': teen.happy, 'sad': !teen.happy}">
  <td>{{teen.name}}</td>
  <td>{{teen.birthday}}</td>
  <td><input type="checkbox" ng-model="teen.happy"/></td>
</tr>
```

- To achieve 2-way data binding, you use ng-model to attach an element to something in the $scope
- As the checkbox is clicked, the underlying structure changes
- The ng-class construct also applies the appropriate class as the underlying model changes
- You can also do read-only data binding using ng-bind instead of ng-model

Filters and Formatters

- There are several built-in filters, which can do either filtering or formatting of data within your client side templates.
- Formatters: currency, date, json, lowercase, number, uppercase

```
<td>{{teen.birthday | date:'fullDate'}}</td>
```

- Filters: limitTo, orderBy

Name	Birthday
Teen One	Monday, January 1, 2001

- Let's take a look at how date formatting can be done.
- Follow along with this Plunker

Watch a $scope variable

- [ ] AngularJS allows us to monitor a $scope variable using $watch
- [ ] Allows you to have a callback fire whenever the value changes
- [ ] It is an alternative to using ng-click or ng-change
- [ ] Replaces jQuery click or change events.
- [ ] Follow along with this Plunker

Watch a $scope variable (code)

```
myApp.controller('PersonCtrl', function ($scope, Adult) {

  $scope.giftTotal = 0;

  $scope.$watch('giftTotal', function(newVal, oldVal) {
    if(newVal === oldVal) { // happens on initial $watch registration
      return;
    }
    console.log('New Val:' + newVal + ' Old Val:' + oldVal);
  });

  // Adults are from REST API (really a static json file)
  $scope.adults = Adult.query();

});
```

- ☐ You can use $watch on a $scope variable to hook into whenever it is changed

Client-side routing

- [ ] Allows you to map URL fragments, using #, to templates and controllers to design single page apps easier.
- [ ] Perfect for perma-linking and allowing browser history to work like the user would expect.
- [ ] Uses ngRoute to accomplish this mapping of URL paths to templates and controllers.
- [ ] Similar function to a Front Controller design pattern that you would use server-side in MVC design.
- [ ] Follow along with this Plunker

Client-side routing (code)

```
/**
 * This is the configuration for the routes
 * Maps a URL fragment to a template and controller
 */
myApp.config(function($routeProvider) {
  $routeProvider.
    when('/adults', {
      templateUrl: 'adult-list.html',
      controller: 'PersonCtrl'
    }).
    when('/adults/:adultId', {
      templateUrl: 'adult-detail.html',
      controller: 'PersonDetailCtrl'
    }).
    otherwise({
      redirectTo: '/adults'
    });
});
```

- You map hash routes to a controller and template
- #/adults is how the path would look in the address bar. (or #/adults/1)

Other Notable Directives...

- ☐ The AngularJS API page has a full list of directives.
- ☐ ngShow and ngHide similar to jQuery .show() and .hide()
- ☐ ngInclude, ngIf, ngSwitch to use a template and manipulate the DOM based on a condition

Interacting with a REST API

```
myControllers.controller('PersonCtrl', function ($scope, Adult) {
  $scope.adults = Adult.query();
});
```

- As seen in other examples, the syntax makes the Asynchronous REST API call “appear” synchronous.
- The call to query() returns an empty object, that is filled back in when the AJAX response returns.
- There are a lot of default methods provided in ngResource, get(), query(), remove(), delete(), save().
- Let’s do a $save, Follow along with this Plunker

Interacting with a REST API (code)

```
<div>Birthday: {{adult.birthday}}</div>
<div>Happy?: <input type="checkbox" ng-model="adult.happy" ng-change="saveAdult(adult)"/></div>
```

```
/**
 * Create the controller for the detail view
 * Using route it is hooked into DOM using ng-view directive
 */
myApp.controller('PersonDetailCtrl', function ($scope, $routeParams, Adult) {
  $scope.adultId = $routeParams.adultId;
  $scope.adult = Adult.get({adultId: $routeParams.adultId});

  $scope.saveAdult = function(adult) {
    adult.$save();
  };
});
```

- ☐ Other than $watch, you have a lot of other hooks into view changes. Here we use ng-change to call a function to issue an AJAX $save to the REST API
- ☐ The argument to the Adult REST API is pulled from the $routeParams, the server is called like /adults/1.json

How Does $watch Work

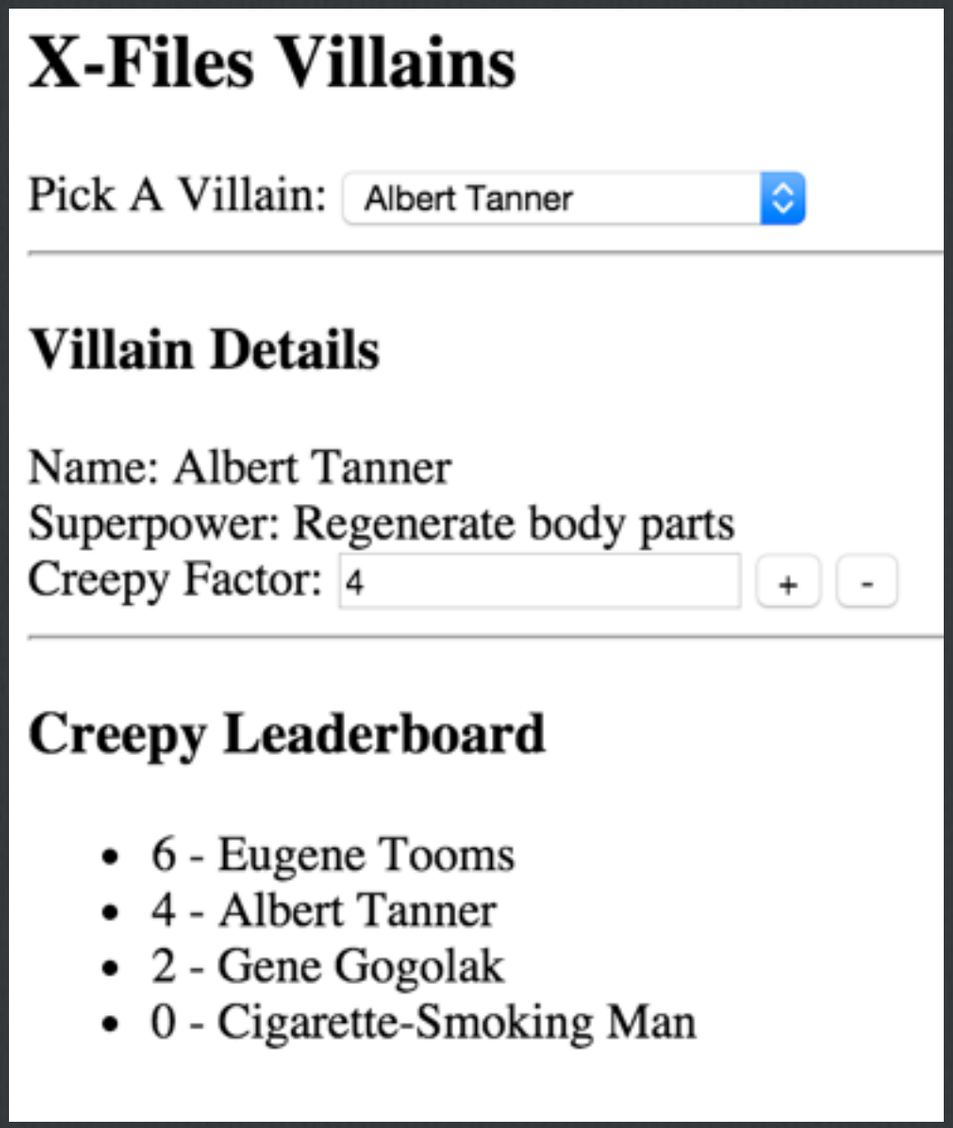
- The whole $watch process is a great design pattern to follow in Web Applications. This is generically called the Observer design pattern.
- It goes along with $digest and $apply, it is a part of the Digest cycle.
- When an $apply is run, this causes the Digest cycle to kick off and compare the variable values, previous to current. This only happens to variables you have bound to.
- This part of AngularJS is likely where you are first exposed to it's inner workings. Why? Custom Directives!

Custom Directives

Let's build a mini-app

X-Files Villains App - Demo

- [ ] X-Files is Baaaaaacck!!!!
- [ ] Vote on how creepy each of the different X-Files villains are.
- [ ] Plunker Link



X-Files Villains App

- [ ] We will design this using one controller, three directives, one for each section of the app

- [ ] Data is all stored on $scope in controller, and passed via directive scope into the directives, where it is modified via 2-way binding.

- [ ] Controller <-> Directive 1 (2 way data binding)

X-Files Villains App Controller

```
module.controller("AppController", function($scope, villainService) {

  $scope.data = {};

  // getting the villains (maybe from an API)
  $scope.data.villains = villainService.query();

  // store the selected villain in the list to view details of
  $scope.data.selectedVillain = $scope.data.villains[0];

});
```

- Villains data is an array of objects

```
module.service('villainService', function() {
  var data = [
    {
      id: "tooms",
      name: "Eugene Tooms",
      superpower: "Body contortion",
      creepy: 0
    },
    {
      id: "smokingman",
      name: "Cigarette-Smoking Man",
      superpower: "Defying death",
      creepy: 0
    },
```

X-Files Villains App
Main Template

```
<body ng-app="app">
  <h1>AngularJS Directive Architecture Example</h1>
  <div ng-controller="AppController">
    <h2>X-Files Villains</h2>
    <villain-list villains="data.villains" selected-villain="data.selectedVillain"></villain-list>
    <hr/>
    <villain-detail villain="data.selectedVillain"></villain-detail>
    <hr/>
    <div>
      <h3>Creepy Leaderboard</h3>
      <villain-leaderboard villains="data.villains"></villain-leaderboard>
    </div>
  </div>
</body>
```

- ☐ See 3 Directives, they accept data.villains and data.selectedVillain, are the two items on the controller scope

X-Files Villains App
Villain List Directive

```
<label>Pick A Villain:</label>
<select ng-options="villain.name for villain in villains" ng-model="selectedVillain"></select>
```

- We use scope to define the attributes
- The '=' is for two way data binding
- templateUrl points to a HTML template
- The ng-model will change selectedVillain on the main controller

```
module.directive('villainList', function() {
  return {
    scope: {
      villains: '=',
      selectedVillain: '='
    },
    templateUrl: 'villains/villainList/villainList.html'
  };
});
```

X-Files Villains App
Villain Detail Directive

- [ ] Villain Detail ng-model shows the creepy attribute

- [ ] Clicking on the buttons calls a function on the $scope of the controller

```
<h3>Villain Details</h3>
<div>Name: {{ villain.name }}</div>
<div>Superpower: {{ villain.superpower }}</div>
<div>
  <span>Creepy Factor:</span>
  <input type="number" ng-model="villain.creepy"/>
  <button type="button" ng-click="moreCreepy(villain)">+</button>
  <button type="button" ng-click="lessCreepy(villain)">-</button>
</div>
```

```
module.directive('villainDetail', function() {
  return {
    scope: {
      villain: '='
    },
    templateUrl: 'villains/villainDetail/villainDetail.html',
    controller: function($scope) {

      // these functions were moved in from the app controller
      // to this directive controller
      $scope.moreCreepy = function(villain) {
        villain.creepy++;
      };

      $scope.lessCreepy = function(villain) {
        villain.creepy--;
      };
    }
  };
});
```

X-Files Villains App
Villain Detail Directive

```
<ul>
  <li ng-repeat="villain in villains | orderBy:'creepy':true">{{ villain.creepy }} - {{ villain.name }}</li>
</ul>
```

```
module.directive('villainLeaderboard', function() {
  return {
    scope: {
      villains: '='
    },
    templateUrl: 'villains/villainLeaderboard/villainLeaderboard.html'
  };
});
```

- Here we show using a filter on an array of objects
- orderBy takes a parameter, the property on the object to use to sort by.

Directive Best Practices

- ☐ Choices to consider
  - ☐ Using Attribute vs. Element
  - ☐ Using proper HTML5/XHTML5 markup
  - ☐ Scope considerations for reusable code
- ☐ See my blog post on AngularJS Directive Best Practices

Directive Best Practices (Forms of directives)

```
<my-dir></my-dir>
<span my-dir="exp"></span>
<!-- directive: my-dir exp -->
<span class="my-dir: exp;"></span>
```

```
<span ng-bind="name"></span> <br/>
<span ng:bind="name"></span> <br/>
<span ng_bind="name"></span> <br/>
<span data-ng-bind="name"></span> <br/>
<span x-ng-bind="name"></span> <br/>
```

- ☐ Ways to reference a directive from within a template.
- ☐ Equivalent examples of an attribute that would match ng-bind.

What I haven't (and won't) cover in detail

- ☐ Form validation
- ☐ Instead of ngResource, you can just use $http for lower level control (closer to jQuery ajax/get)
- ☐ Promises (Deferred API)
- ☐ Writing tests + how the $injector works
- ☐ Animations
- ☐ Lots more…

Why Use AngularJS?

Now we know what AngularJS can do.
Why should we integrate it into our web application?

Why should you use AngularJS in your next web app?

- ☐ Encourages good web app front-end design practices
  - ☐ Model as the source of truth
  - ☐ Using classes for style not functionality
  - ☐ Dependency Injection core to framework to have code that is test-ready
  - ☐ Use client-side objects that are similar to server-side objects
  - ☐ Easy to hook up to REST API to have server just providing data and HTML/templates

- [ ] Less code to write, recall the jQuery vs. AngularJS example
- [ ] Creating directives that encourage re-use and easy to be shared with others
- [ ] Easy to collaborate with other developers by using object-oriented design principles, reusable components, and focus on testability
- [ ] Client-side templating
- [ ] Does not depend on jQuery, so you don't need to include both.

Weaknesses of AngularJS

- ☐ No server-side templating (supposedly version 2.0).
- ☐ No easy way to switch out to use a different templating engine
- ☐ Putting functionality embedded in HTML may not feel like good enough separation of presentation and code.
- ☐ SEO for public-facing web apps is difficult to achieve due to no server-side templating
  - ☐ Using PhantomJS to create snapshots and save, then use #! in URL: link
- ☐ Have to be careful of over-$watching. You can watch all properties of every object if you really want to.

Angular 1.5 vs. React

- ☐ React by itself is not a fair comparison with AngularJS. React is typically paired with something like Flux or Redux to be able to do full apps.
- ☐ AngularJS is all about 2 way data binding. React approach is about managing state and one way data flow.
- ☐ AngularJS uses watches and data binding to know when to update DOM. React uses a Virtual DOM to improve DOM updating efficiency by only applying changes.
- ☐ React encourages using immutable data structures, resulting in operations one way.
- ☐ AngularJS uses its own templating system, React can use JSX templates that exist right alongside code.
- ☐ AngularJS has a lot more contributed libraries and UI components because it has been around longer.

Angular 1.5 vs. Angular 2

- [ ] Angular 2 is currently in beta.
- [ ] Angular 2 apps will be built entirely of components, like React. Directives in 1.x are the closest thing we have to what a component looks like.
- [ ] Angular 1.x release are adding small steps you can take to make the upgrade easier. Angular 1.5 even added a component type, that is basically a more restrictive directive.
- [ ] Will be able to use server side templating, able to be used easier in all types of apps in more Javascript environments.
- [ ] Most examples are in TypeScript, but will also be able to use ES2015.
- [ ] Just like React, encourages using explicit data operations, resulting in operations one way and in general improving app performance over Angular 1.

Frontend Web Dev Trends

- [ ] Frontend separate from backend, using API for communication. Enables web apps, and mobile apps, to be developed without rethinking architecture. Ready to scale and allows developer specialization.
- [ ] Movement towards using a single app state (React+Redux and Om)
- [ ] Improving rendering performance by minimizing direct DOM interaction (React and Angular 2)
- [ ] Javascript mobile apps using a hybrid approach of native UI components and WebView for improved performance over WebView-only (React Native)

Where do you go next?

- ☐ Basic Tutorial on AngularJS site
- ☐ Developer Guide on AngularJS site
- ☐ Angular 2 Site - angular.io
- ☐ ng-conf 2015 videos are on YouTube
- ☐ Paid video-based training at egghead.io
- ☐ See all the code from this presentation, and more, at My Plunker Page
- ☐ Build Something Great!

Thanks!

Jeremy Zerr

Blog: http://www.jeremyzerr.com
LinkedIn: http://www.linkedin.com/in/jrzerr
Twitter: http://www.twitter.com/jrzerr